OFFICE COPY

UNITED STATES DISTRICT COURT
SOUTHERN DISTRICT OF NEW YORK

ALADDIN KNOWLEDGE SYSTEMS, LTD.,

Plaintiff,

-against-

FINJAN, INC., FINJAN SOFTWARE, LTD., and FINJAN SOFTWARE, INC.

Defendants.

Civil Action No. 08-cv-5611 (RJH)

JURY TRIAL DEMANDED

RECEIVED
AUG 22 2008
U.S.D.C. S.D. N.Y.
CASHIERS

AMENDED COMPLAINT FOR PATENT INFRINGEMENT AND JURY DEMAND

Plaintiff Aladdin Knowledge Systems, Ltd. ("Aladdin") complains against Defendants Finjan, Inc., Finjan Software, Ltd., and Finjan Software, Inc. as follows:

THE PARTIES

1. Aladdin is a corporation organized and existing under the laws of Israel, with its corporate headquarters at 35 Efal St., Kiryat Arye, Petach Tikva, Israel 49511.

2. On information and belief, Finjan, Inc. ("Finjan, Inc.") is a Delaware corporation with its principal place of business at 2025 Gateway Place, Suite 180, San Jose, California 95110. On information and belief, as of the time this action was filed, Finjan, Inc. had corporate offices in New York at 405 Lexington Avenue, 35th Floor, New York, NY 10174.

3. On information and belief, Finjan Software, Inc. ("Finjan Software") is a Delaware corporation with its principal place of business at 2025 Gateway Place, Suite 180, San Jose, California 95110. On information and belief, as of the time this action was filed, Finjan Software had corporate offices in New York at 405 Lexington Avenue, 35th Floor, New York, NY 10174.

4. On information and belief, Defendant Finjan Software, Ltd. ("Finjan Israel") is a corporation organized and existing under the laws of Israel, with its principal place of business at Hamachshev St. 1, New Industrial Area, Netanya, Israel 42504.

5. Aladdin owns United States Patent No. 7,013,483 (the "'483 Patent"), entitled "Method for Emulating an Executable Code In Order to Detect Maliciousness," and United States Patent No. 7,386,884 (the "'884 Patent"), entitled "Method for Preventing Activation of Malicious Code." Finjan Israel, Finjan, Inc., and Finjan Software (collectively referred to herein as "Defendants") make, use, sell, import and/or offer for sale computer security software that infringes both of these patents.

JURISDICTION AND VENUE

6. This Court has subject matter jurisdiction over this action as it arises under the federal patent laws of the United States of America. 28 U.S.C. §§ 1331 and 1338(a).

7. The Court has personal jurisdiction over Defendants because, on information and belief, Defendants have systematic and continuous contacts with the State of New York and with this judicial district such that the exercise of jurisdiction over Defendants satisfies the requirements of C.P.L.R §§ 301, 302 and does not offend traditional notions of fair play and substantial justice.

8. This Court has jurisdiction over Finjan, Inc. because, on information and belief, it has made, used, sold, offered for sale, and/or imported, and is currently making, using, selling, offering for sale, and/or importing, products and related services within or into New York and this judicial district that infringe Aladdin's patents asserted in this action. Moreover, Finjan, Inc. has a continuous and systematic presence in the State of New York and this judicial district, and is doing business in New York with a fair measure of permanence and continuity. For example, according to records maintained on the Secretary of State of New York's website, Finjan, Inc. is an active foreign corporation doing business in New York, and has designated the Secretary of

State of New York as its agent for service of process. Furthermore, at least as of June 20, 2008, Finjan, Inc. maintained an office at 405 Lexington Avenue, 35th Floor, New York, NY 10174. On information and belief, Finjan, Inc. has at least three to eight active customers in New York (who it is systematically and continuously servicing) with many more New York customers and customer prospects who are being solicited or serviced. On information and belief, Finjan, Inc. sells products (including infringing products) in New York, maintains a technical support center in New York, and routinely sends employees to New York to solicit business, including but not limited to Finjan's infringing products. For example, on information and belief, in or around March 2008, Finjan, Inc. employees attended a meeting at the New York offices of Merrill Lynch where they offered Finjan products for sale. In addition, Finjan, Inc.'s Chief Technology Officer (who, on information and belief, is also an officer of Finjan Israel), gave a sales presentation at the 11th Annual New York State Cyber Security Conference on June 5, 2008 in Albany and at the Ziff Davis Enterprise Security Summit trade show on March 14, 2007 at the Marriott hotel in New York City and spoke *inter alia* about the advantages of using Finjan products (including infringing products) with the purpose of selling such products to New York residents. Further, Finjan, Inc. currently specifically lists on its website sales contacts in its California office for Finjan products in New York. On information and belief, Finjan, Inc. also has strategic and technological alliances with New York corporations. For example, on information and belief, Finjan is in an alliance with International Business Machines Corporation, a New York corporation. In addition, on information and belief, at least one member of the board of directors of Finjan, Inc., Adam Fisher, resides in New York.

9. This Court has jurisdiction over Finjan Software because, on information and belief, Finjan Software has made, used, sold, offered for sale, and/or imported, and is currently making,

using, selling, offering for sale, and/or importing, products and related services within or into New York and this judicial district that infringe Aladdin's patents asserted in this action. This Court also has jurisdiction over Finjan Software because, on information and belief, Finjan Software has a continuous and systematic presence in the State of New York and this judicial district, and is doing business in New York with a fair measure of permanence and continuity. For example, on information and belief, Finjan Software is the author of several white papers and product brochures regarding the Finjan computer security products that are at issue in this action. On information and belief, these white papers and product brochures are distributed by Finjan Software to New York residents with the purpose of selling Finjan computer security products to New York residents.

10. This Court has jurisdiction over Finjan Israel because, on information and belief, it has made, used, sold, offered for sale and/or imported, and is currently making, using, selling, offering for sale and/or importing, products and services within or into New York and this judicial district that infringe Aladdin's patents asserted in this action. This Court also has jurisdiction over Finjan Israel because, on information and belief, Finjan Israel is doing business in New York and has a continuous and systematic presence in New York. This Court also has jurisdiction over Finjan Israel because, on information and belief, Finjan Israel has committed and continues to commit tortious acts out of state that have caused an injury within New York, and Finjan Israel should reasonably expect the acts to have consequences in New York. On information and belief, Finjan Israel also derives substantial revenue from international commerce.

11. On information and belief, all Finjan products sold in the United States, including those sold to customers in New York and in this judicial district and that infringe the patents

asserted in this action, originate from Finjan Israel. On information and belief, Finjan Israel employees travel to New York at least ten times a year to sell Finjan products, meet with customers and potential customers, market Finjan products at trade shows, conferences and seminars, and assist in the administration of Defendants' infringing activities in New York. On information and belief, prior to November 2007, Defendants' Vice President of Sales in the United States, whose sales region includes New York, was under the direction of and reported to the CEO of Finjan Israel. On information and belief, Finjan Israel maintains a New York based marketing manager who markets Finjan products (including but not limited to infringing products) to potential customers in New York and who reports to the CTO of Finjan Israel. On information and belief, beginning prior to November 2007 and continuing to the present time, Finjan Israel employees have attended sales meetings in New York and in this judicial district and have set the agenda for and lead those sales meetings in New York and elsewhere in the United States. On further information and belief, when in New York to sell Finjan products to New York customers, Finjan Israel employees represent that they are in New York on behalf of Finjan Israel and give business cards to customers with contact information in Israel. On information and belief, the sales meetings in New York were and are for the purpose of offering for sale Finjan products that infringe Aladdin's patents asserted in this action. For example, on information and belief, one recently departed Finjan Israel employee attended at least three or four sales meetings in New York (including in this judicial district) on behalf of Finjan Israel including one in around March 2008 at the New York City offices of Merrill Lynch while he was a Finjan Israel employee. On information and belief, during these meetings, this employee represented that he was a representative of Finjan Israel and was in New York (including in this judicial district) on behalf of Finjan Israel. Further, on information and belief, in at least two of his meetings in New York this employee set the agenda for and ran the sales meeting, including directing any of Defendants' employees who were in attendance. On information and belief, Defendants' CTO (who is the CTO of Finjan Israel), Yuval Ben-Itzhak, who resides in Israel and is paid by Finjan Israel, has presented business cards at sales meetings in New York that identified him as a representative of Finjan Israel and that listed contact information in Israel. On information and belief, Defendants' CTO controls marketing and research and development of all Finjan products including marketing directed to New York. On information and belief, the General Manager of Finjan Israel, Marcio Lampert, is the Vice President of International Sales for Defendants. In addition, on information and belief, Finjan Israel maintains a technical support center in Israel that provides technical support services for all of Defendants' customers, including support services to and direct contact with Defendants' customers in New York (including in this judicial district).

12. On further information and belief, Finjan Israel is the headquarters for research and development for all Finjan products, including those sold within the United States, including New York and in this judicial district. Finjan Israel is also the headquarters, on information and belief, for the Malicious Code Research Center, which supports the development of Finjan products, provides information on Finjan products to Finjan customers including those in New York, hosts customer seminars, writes research papers distributed on Defendants' website and in New York, and sends employees to lecture at various locations and trade shows including within New York, this judicial district, and the United States. For example, on information and belief, Iftach Amit while a Finjan Israel employee, presented at the Integralis USA PCI Compliance Roadshow, on April 17, 2007 in New York City and spoke *inter alia* about the advantages of using Finjan products. Similarly, Yuval Ben-Itzhak, on information and belief, presented at the

11th Annual New York State Cyber Security Conference on June 5, 2008 in Albany and at the Ziff Davis Enterprise Security Summit trade show on March 14, 2007 at the Marriott hotel in New York City and spoke *inter alia* about the advantages of using Finjan products. On information and belief, these presentations were made in New York for the purpose of selling infringing Finjan products to New York residents.

13. The Court also has jurisdiction over Defendants because, on information and belief, Defendants are interrelated entities and are mere departments of one another. On information and belief, Finjan, Inc., Finjan Software and Finjan Israel disregard the separate corporate existence of each entity; are commonly owned; are financially interdependent; fail to observe corporate formality; share common officers, managers and directors; and uniformly control marketing and operational policies. For example, on information and belief, officers are freely transferred between Finjan, Inc. and Finjan Israel, which have at least three overlapping officers: Yuval Ben-Itzhak, the Chief Technology Officer; Ron Kreitsman, the Chief Financial Officer; and Marcio Lampert, the General Manager and Vice President of International Sales. On information and belief, prior to November 2007, the Vice President of Sales for North America, an employee of Finjan, Inc., reported to and was under the direction of the CEO and President of Finjan Israel. On information and belief, the current CEO of Finjan, Inc. and Finjan Software was recruited and hired by Mr. Ben Itzhak and Mr. Lampert, both officers and employees of Finjan Israel. On information and belief, the marketing budget of Finjan Inc. is controlled by the CTO of Finjan Israel. On information and belief, sales training for employees of Finjan, Inc. is managed and directed by employees of Finjan Israel.

14. Further, on information and belief, Defendants' employees do not distinguish between Finjan Israel, Finjan, Inc. and Finjan Software. Defendants' website and press releases

make no distinction between any of the Defendants and refer to them interchangeably and as a single entity. For example, the Finjan website alternatively lists Finjan, Inc. or Finjan Software as the copyright holder of various webpages and lists Finjan Israel as an office of Finjan, Inc. and Finjan Software — not as a separate entity. Also, a newsletter from July 2006 makes no distinction between the "Finjan" awarded a patent and the "Finjan" that opened an office in New York, although, on information and belief, the former was Finjan Israel and the later was Finjan, Inc.

15. Venue is proper in this judicial district because, on information and belief, Defendants are subject to personal jurisdiction, do business, and have committed acts of infringement in this judicial district. 28 U.S.C. §§ 1391(b), (c), and 1400(b).

INFRINGEMENT OF U.S. PATENT NO. 7,013,483

16. On March 14, 2006, the United States Patent & Trademark Office duly and legally issued United States Patent No. 7,013,483 (the "'483 Patent"), entitled "Method for Emulating an Executable Code In Order to Detect Maliciousness." Aladdin is the owner by assignment of the '483 Patent, a true and correct copy of which is attached as Exhibit A.

17. On information and belief, Defendants have infringed and continue to directly and/or contributorily infringe (literally and under the doctrine of equivalents) the '483 Patent by, among other things, making, using, offering for sale, and/or selling, or inducing others to make, use, offer for sale, and/or sell products including computer security products within the United States that are within the scope of one or more claims of the '483 Patent. Defendants are therefore liable for infringement of the '443 Patent. 35 U.S.C. § 271.

18. Defendants' acts of infringement have caused and are continuing to cause monetary damage to Aladdin in an amount to be determined at trial. In addition to monetary damages,

Defendants' infringement has caused and will, unless enjoined, continue to cause irreparable harm to Aladdin's business.

19. On information and belief, Defendants' infringement of the '483 Patent was and has been at all relevant times willful and deliberate, such that Aladdin is entitled to enhanced damages as well as attorneys' fees and costs incurred in prosecuting this action. 35 U.S.C. §§ 284 and 285.

INFRINGEMENT OF U.S. PATENT NO. 7,386,884

20. On June 10, 2008, the United States Patent & Trademark Office duly and legally issued United States Patent No. 7,386,884 (the "'884 Patent"), entitled "Method for Preventing Activation of Malicious Code." Aladdin is the owner by assignment of the '884 Patent, a true and correct copy of which is attached as Exhibit B.

21. On information and belief, Defendants have infringed and continue to directly and/or contributorily infringe (literally and under the doctrine of equivalents) the '884 Patent by, among other things, making, using, offering for sale, and/or selling, or inducing others to make, use, offer for sale, and/or sell products including computer security products within the United States that are within the scope of one or more claims of the '884 Patent. Defendants are therefore liable for infringement of the '884 Patent. 35 U.S.C. § 271.

22. Accordingly, Aladdin is entitled to recover damages in an amount adequate to compensate it for infringement, but in no event less than a reasonable royalty. In addition to monetary damages, Defendants' infringement has caused and will, unless enjoined, continue to cause irreparable harm to Aladdin's business.

23. On information and belief, Defendants' infringement of the '884 Patent was and has been at all relevant times willful and deliberate, such that Aladdin is entitled to enhanced

damages as well as attorneys' fees and costs incurred in prosecuting this action. 35 U.S.C. §§ 284 and 285.

PRAYER FOR RELIEF

WHEREFORE, Aladdin prays for the following relief:

1. Judgment in favor of Aladdin finding that Defendants have directly and/or indirectly infringed literally or under the doctrine of equivalents the '483 Patent and the '884 Patent;

2. A preliminary and permanent injunction enjoining Defendants and their officers, directors, agents, servants, affiliates, employees, divisions, branches, subsidiaries, parents, and all others acting in concert or privity with any of them, from any further infringement of the '483 Patent and the '884 Patent;

3. An award of damages, costs and expenses;

4. An award of pre-judgment and post-judgment interest;

5. An award of enhanced damages pursuant to 35 U.S.C. § 284;

6. An award of attorneys' fees and costs pursuant to 35 U.S.C. § 285 or as otherwise permitted by law; and

7. For all other relief to which the Court may deem just and proper.

DEMAND FOR JURY TRIAL

Pursuant to Rule 38 of the Federal Rules of Civil Procedure, Aladdin requests a trial by jury of any and all issues so triable by right.

DATED: New York, New York
August 22, 2008

QUINN EMANUEL URQUHART OLIVER & HEDGES, LLP

By: Mark Baker

Claude M. Stern (*pro hac vice*)
claudestern@quinnemanuel.com
Evette D. Pennypacker (*pro hac vice*)
evettepennypacker@quinnemanuel.com

555 Twin Dolphin Drive, Suite 650
Redwood Shores, California 94065
(650) 801-5000

Mark D. Baker
markbaker@quinnemanuel.com
Joshua Furman
joshuafurman@quinnemanuel.com

51 Madison Avenue, 22nd Floor
New York, New York 10010-1601
(212) 849-7000

Attorneys for Plaintiff

Exhibit A

US007013483B2

(12) **United States Patent**
Cohen et al.

(10) **Patent No.: US 7,013,483 B2**
(45) **Date of Patent: Mar. 14, 2006**

(54) METHOD FOR EMULATING AN EXECUTABLE CODE IN ORDER TO DETECT MALICIOUSNESS

(75) Inventors: **Oded Cohen**, Tivon (IL); **Inbal Meir**, Haifa (IL); **Yanki Margalit**, Ramat-Gan (IL); **Dany Margalit**, Ramat-Gan (IL)

(73) Assignee: **Aladdin Knowledge Systems Ltd.**, Tel Aviv (IL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 286 days.

(21) Appl. No.: **10/335,871**

(22) Filed: **Jan. 3, 2003**

(65) **Prior Publication Data**

US 2004/0133796 A1 Jul. 8, 2004

(51) **Int. Cl.**
H04L 713/20 (2006.01)

(52) **U.S. Cl.** **726/25**; 726/23; 726/24; 713/165; 713/167; 713/188

(58) **Field of Classification Search** 713/188, 713/200–201, 167, 165; 380/4, 3
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,398,196	A	*	3/1995	Chambers	714/28
5,842,002	A	*	11/1998	Schnurer et al.	703/21
5,978,917	A	*	11/1999	Chi	713/201
6,035,423	A		3/2000	Hodges et al.	
6,067,410	A	*	5/2000	Nachenberg	703/28
6,269,456	B1		7/2001	Hodges et al.	
6,357,008	B1	*	3/2002	Nachenberg	713/200
2002/0013910	A1		1/2002	Edery et al.	
2002/0073330	A1		6/2002	Chandnani et al.	

FOREIGN PATENT DOCUMENTS

WO	WO01/88673	11/2001
WO	WO02/37740	5/2002

OTHER PUBLICATIONS

http://livedocs.macromedia.com/colfusion/6/Developing__coldFusion__MX__Applications__with__CFML/queryDB6.htm.*
Pleszkoch et al., "Improving Network System with Function Extraction Technology for Automated Calculation of Program Behavior", Proceeding of the 37th Hawaii International Conference on System Science, 2004.*
http://www.cgisecurity.com/owasp/html/ch11s04.*

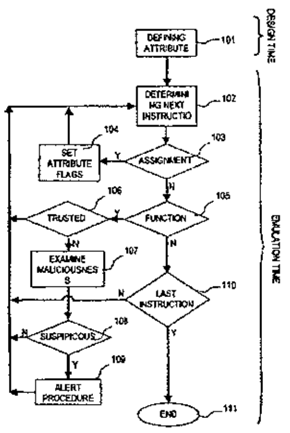
* cited by examiner

Primary Examiner—Gregory Morse
Assistant Examiner—Tongoc Tran
(74) *Attorney, Agent, or Firm*—Mark M. Friedman

(57) **ABSTRACT**

The present invention is directed to a method for emulating an executable code, whether it is a human-readable code (e.g., macro and script) or a compiled code (e.g. Windows executable). At the design time, one or more content attributes are defined for the variables of the code. A content attribute indicates a property with relevance to maliciousness, e.g. Windows directory, a random value, ".EXE" at the right of a string, etc. A content attribute may be implemented, for example, by a flag. Also defined at the design time, is a list of malicious states, where a malicious state comprises at least the combination of a call to a certain system function with certain content, as the calling parameter(s). When emulating an assignment instruction, the attributes of the assigned variable are set according to the assigned content. When emulating a mathematical operator, a content mathematics is also applied. When emulating a function call, the current state (i.e. the function identity and the calling content and values) is compared with the pre-defined malicious states, and if at least one malicious state corresponds, then the maliciousness of the code is determined.

24 Claims, 1 Drawing Sheet



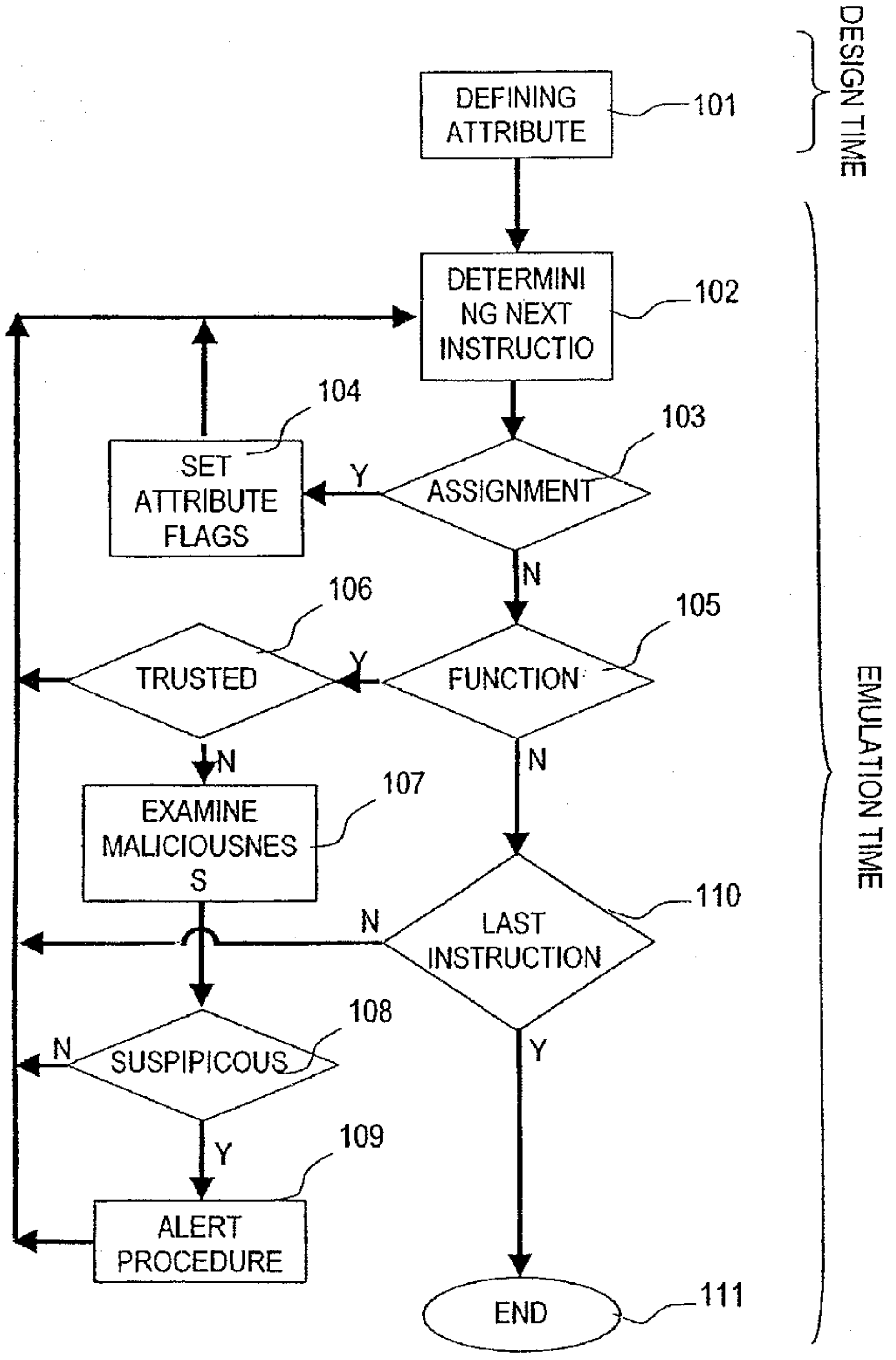

DEFINING ATTRIBUTE
101
DESIGN TIME
DETERMINING NEXT INSTRUCTIO
102
ASSIGNMENT
103
Y
SET ATTRIBUTE FLAGS
104
N
FUNCTION
105
Y
TRUSTED
106
N
EXAMINE MALICIOUSNESS
107
N
LAST INSTRUCTION
110
N
SUSPIPICOUS
108
Y
ALERT PROCEDURE
109
Y
END
111
EMULATION TIME

Fig. 1

METHOD FOR EMULATING AN EXECUTABLE CODE IN ORDER TO DETECT MALICIOUSNESS

FIELD OF THE INVENTION

The present invention relates to the field of detecting presence of virus and other kinds of malicious code by emulation.

BACKGROUND OF THE INVENTION

Emulation is one of the methods of detecting viral activity. The emulator creates a virtual environment under which the code of the tested sample is examined instruction-by-instruction. The viral activity is detected by correlating the changes with a viral behavior.

Emulation has several advantages in comparison to other methods of detecting and/or preventing viral damages, since it helps to detect polymorphic viruses, encrypted viruses and unknown viruses, which cannot be detected by other methods.

A programmer can indicate the maliciousness of a source code just by reviewing the code. For example, if he identifies a loop that gets the name of the files of a directory and deletes them, he can assume that the code is malicious. Contrary to a human programmer, an emulator, which is a machine, cannot indicate viral code just by reviewing the code, but it has to execute the code while comparing the caused changes with known patterns. In other words, the emulator, being a machine, is absent of human intelligence.

It is an object of the present invention to provide a method for emulating an executable code in order to detect maliciousness, which is more "intelligent" than known in the prior art.

Other objects and advantages of the invention will become apparent as the description proceeds.

SUMMARY OF THE INVENTION

The present invention is directed to a method for emulating an executable code, whether it is a human-readable code (e.g., macro and script) or a compiled code (e.g. Windows executable). At the design time, one or more content attributes are defined for the variables of the code. A content attribute indicates a property with relevance to maliciousness, e.g. Windows directory, a random value, ".EXE" at the right of a string, etc. A content attribute may be implemented, for example, by a flag. Also defined at the design time, is a list of malicious states, where a malicious state comprises at least the combination of a call to a certain system function with certain content, as the calling parameter(s). When emulating an assignment instruction, the attributes of the assigned variable are set according to the assigned content. When emulating a mathematical operator, a content mathematics is also applied. When emulating a function call, the current state (i.e. the function identity and the calling content and values) is compared with the predefined malicious states, and if at least one malicious state corresponds, then the maliciousness of the code is determined.

The system function can be, for example, a part of the programming language of the executable, a function provided by the operating system under which the code is to be executed, a function provided by a linked package of routines.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood in conjunction with the following figure:

FIG. 1 schematically illustrates a method for emulating an executable, in order to detect malicious code.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The term Macro in conjunction with computers, refers herein to an executable code embedded within another entity, like a document or executable code.

The term Script in conjunction with computers, refers in the art to a program built into the HTML code of a Web page that work on almost all Internet applications. Scripts can also be used with a variety of applications that support their scripting syntax. VBScript and JavaScript are examples of scripting languages.

Macros and scripts are "non-compiled executables", i.e. programs wherein their executable code is the source code, contrary to "compiled executables" which are converted to machine-code before execution. ASP and XML are other examples of non-compiled executables.

Due to their human readability, non-compiled executables are exploited by vandals (malicious executables written into scripts and macros). They can do almost anything—infect a system with Trojans, modify files, cause DoS attacks, or spread malicious code using Microsoft Networks or Microsoft Outlook. Macro-viruses that attack Microsoft Word work by copying themselves into the global environment—usually by modifying NORMAL.DOT. Unfortunately, the enhanced power of software applications also gives virus writers the potential to cause more damage.

It should be noted that the examples herein are presented in a human-readable programming code, although they apply to both compiled and non-compiled code. Moreover, the examples are presented in a schematic programming language, rather than an existing programming language.

There are some programming tools and computer resources commonly used by malicious code. For example, the random generator, which is a programming tool provided by almost any operating system and almost any programming language, is commonly used by viruses to prevent the repetition of the same code when duplicating itself, thereby hardening the detection of the malicious code by bit-pattern (also known as "virus signature") scanning. A human-readable code, as a program written in a programming language, makes use of variables. Since it is plain text, the mechanism that duplicates the malicious code can generate random names for its variables, to add non-effective code comments between its effective code lines, etc. The non-effective code may also be generated randomly, thus using the random generator.

The real time clock, which is a programming-tool provided by any operating system and almost any programming language, is another example of a virus-commonly used programming tool, since it enables a late-trigger of the virus code, and also because it can be used as a substitute to the random generator. Moreover, some of the viruses have a late trigger, thereby enabling them to be widely propagated before causing the damage, and consequently their discovery. The late trigger is tuned to a future date, which involves use of the real time clock of the host machine.

Another example of a content used by malicious code in Windows operating system is the Windows folder, which stores Windows system files. A virus may try to change one of the existing files in the Windows folder, create a file with a random name, etc.

A combination of some of the contents described above may also indicate the presence of malicious code within an executable. For example, creating a new file having a random name in the Windows directory can be due to a virus. However, a more accurate indication of a malicious code can be obtained if this is an executable file. Thus, if the created file has an EXE extension, which is the extension of Windows executables, or VBS, which is the extension of Visual Basic Script, the executable can be classified as malicious.

According to a preferred embodiment of the present invention, a "malicious state" comprises at least executing a certain system function with certain content as its parameter (s). Thus, at least two elements are involved—a certain system function and certain content as the parameter(s).

It should be noted that the term "content" of a variable refers herein also to its value. Nevertheless, in some cases the value has been also mentioned.

A state can be indicated as malicious if it corresponds to at least one pre-defined malicious state. Two states "correspond" if their elements (i.e. function identity and the calling content) are equal or at least "similar (e.g., at least certain percentage of their content/value is equal). Maliciousness can be defined also by tracking the history of the called functions, e.g., if certain functions have been called with certain content, or even without regard-ness to their content.

Identifying a call to a certain function is quite obvious, and almost any compiler parser performs it. However, the concept of dealing with the content of a parameter rather than its value is novel.

For example, the parameter when calling the following function is a random number:

```
MyVariable=Random( )

MyVariable=MyVariable+1

Call MyFunction(My_Variable)
```

Where Random( ) is a random number generator.

The first time MyFunction is performed, its parameter may be, e.g., 0.443213, and the next time it can be, e.g., 0.332455. Thus, from analyzing the text it is not possible to indicate that the parameter has a random nature.

According to a preferred embodiment of the invention, the content of a variable can be indicated as follows:

Each variable of the executable is characterized by several attributes (e.g. flags), which points out on its content (referred herein as content attributes). For example:

- a flag to indicate if the variable has been assigned with the Windows folder name;
- a flag to indicate if the variable has been assigned with a random value;
- a flag to indicate if the variable has been assigned with the real time;
- a flag to indicate if the variable has been assigned with the value ".EXE", which is the extension of a type of executable file under the Windows operating system;

and so forth.

While emulating the execution of an executable, the following instructions (for example) are performed:

```
File_name=Windows_directory( )+Random( )+0.2+".EXE"

Create_file(File_name)
```

Where: Windows_directory( ) is a system function that returns a string of the Windows directory, and Random returns a numeric random integer.

In this example, the Windows folder flag, the Random flag and the EXE flag of the variable File_name are turned on.

It should be noted that

```
File_name=Windows_directory( )+Random( )+0.2+".EXE"
```

actually comprises several instructions:

Executing the Random function;

Executing the Windows_directory function;

Mathematical addition;

String addition.

It should also be noted that while some programming languages convert a numeric value to a string value and vice versa when it is required, some compilers have no such ability. Thus, assuming the function Str converts a numeric value to a string, in this case the example should look like:

```
File_name=Windows_directory( )+Str(Random( )+0.2)+".EXE"
```

There is no point in considering assigning a value to variables as malicious operation; however without tracking the history of the assignments to a variable, its content cannot be determined. Thus, the examining stage should be at the point where specific system functions which may cause damage to the computer are called, e.g. a function that creates a file, a function that changes the content of a file, and so forth.

The following instruction also creates an executable file with a random name in the Windows directory:

```
Create_file(Windows_directory( )+Random( )+".EXE")
```

However, instead of using a variable as the parameter of the function Create_file, the name of the file is the parameter. Actually, this is equivalent to creating a temporary variable, and assigning a value to the temporary variable just before the execution of said function:

```
Temp_parameter=Windows_directory( )+Random( )+".EXE"

Call Create_file(Temp_parameter)
```

As mentioned above, according to the present invention, the variables comprise content attributes in addition to the value. Thus, there is a question as to how the content is influenced by mathematical operations. For example:

```
MyValue=123+First_var+Second_var
```

Where, for example, the Real time flag of the variable First_var is turned on and the EXE flag of the variable Second_var is turned on. It is obvious that in this case the content result of the assignment is turning on the Real time flag and the EXE flag of the variable MyValue. Thus, the emulator should be provided with "content mathematics" rules. Such rules, which for example comprise a commutative law of addition, an associative law of addition and so forth, should also take into consideration comparison statements and other features supported by the emulated programming language. Of course it is desired to emulate the values as well, in addition to emulating the content.

Another example:

```
Var_A=GetAddressBook( )

Var_B=Var_A->Count( )
```

Where Var_A has the content type "address book object".

When the operator "->Count( )" is operated on an "address book object", it returns not just a numeric value, but a numeric value with the content attribute "number of entries in address book", thus adding the "number of entries in address book" to the variable Var_B.

Another example:

```
Var_A=GetAddressBook( )

Var_C=Var_A[2]
```

After the assignment to Var_C, its value will be the second entry of the address book which has been assigned to Var_A. The content attributes of Var_A may be something like "address book database", and the content attribute of Var_C may be something like "address book entry".

FIG. **1** schematically illustrates a method for emulating an executable, in order to detect malicious code.

At step **101**, which is the design step, a group of malicious states is defined. A malicious state is the combination of a call to a certain system function with certain content. This is a preliminary stage, which is performed at the anti-virus firm, as a part of the anti-virus emulator design. Of course the list of the content attributes and/or functions can reside on a file, and modified later (whether at the anti-virus firm residence or at the installation site).

At the next step **102**, which is performed at the emulation time (i.e. run time), the next instruction to be performed is determined.

At the next step **103**, the type of the instruction is examined. If it is an assignment instruction, then the flow continues with step **104**, otherwise the flow continues with step **105**.

At step **104**, the instruction is analyzed in order to determine the content attribute(s) of the assigned variable to be set, and the corresponding values are set, and from there, the flow returns to step **102**.

At step **105**, if the type of the instruction is a function call, then the flow continues with step **106**; otherwise the flow continues with step **110**.

From step **106**, if the function is pre-determined as trusted, then the flow continues to step **102**; otherwise flow continues to step **107**.

At step **107** the maliciousness is examined by comparing the current state (i.e. the calling function with the content attributes of the parameters) with the malicious states that have been pre-defined in step **101**. Determining the maliciousness of a code can be indicated if the current state corresponds to at least one of the pre-defined malicious states.

From step **108**, if the instruction seems to be suspicious according to the examination carried out at step **107**, then the flow continues with step **109**, where an alert procedure is performed, and from there to step **102**.

From step **110**, if the last tested instruction is the last instruction (e.g. a Stop instruction), then the emulation stops, otherwise the flow continues with step **102**.

Usually, the emulation continues until the last instruction is performed, and then the results of the emulation are examined. This way, more details are gathered and the maliciousness of the code can be determined more precisely. However, the alert procedure can suspend further execution, thereby shortening the emulation time.

As known to a person of ordinary skill in the art, a function can be declared such that a parameter is transferred by its address or by its value. In case of transferring a value, a temporary parameter is created and its address is transferred as a parameter, like in the other case. In both cases when emulating the code, the variables, whether permanent or temporary, also comprise the content attributes.

It should be noted that the term "function" also refers herein to procedures, software interrupts and so forth.

The emulation can be executed at the user's machine, at the gateway to a network and so forth. For example, while a user may install an emulator to check every incoming HTML file while browsing the Internet/Intranet, an organization may install such an emulator at the gateway to its network, at the mail server and so forth.

Those skilled in the art will appreciate that the invention can be embodied by other forms and ways, without losing the scope of the invention. The embodiments described herein should be considered as illustrative and not restrictive.

What is claimed is:

1. A method for emulating an executable code, said method comprising:

pre-defining at least one content attribute having relevance to maliciousness, for at least one variable of said executable code;

pre-defining at least one malicious state, being a call to a certain system function with certain content and/or value;

upon emulating an assignment instruction, setting the content attributes of the assigned variable according to the assigned content, and the value of the assigned variable;

upon emulating a mathematical operator, employing content mathematics; and upon emulating a function call, checking that the current state, being the combination of the function identity with the content and/or value of the calling parameters, corresponds to at least one of the pre-defined malicious states, thereby indicating the maliciousness of said code.

2. The method according to claim **1** , wherein said at least one content attribute is selected from a group comprising: a programming flag, a Boolean value, a numeric value, and a string value.

3. The method according to claim **1**, wherein said executable includes compiled code.

4. The method according to claim **1**, wherein said executable includes human-readable code.

5. The method according to claim **3**, wherein said compiled code is selected from the group consisting of Windows EXE, Windows DLL, Windows OCX, Linux executables, Solaris executables.

6. The method according to claim **4**, wherein said human-readable code is selected from the group consisting of macro, script, VBScript, JavaScript, BAT, PHP, and ASP.

7. The method according to claim **1**, wherein at least one said checking includes testing an equality of at least one element selected from the group consisting of an identity of said function said content of said calling parameters, and said value of said calling parameters.

8. The method according to claim **1**, wherein at least one said checking includes testing a similarity of at least one element selected from the group consisting of an identity of said function, said content of said calling parameters, and said value of said calling parameters.

9. The method according to claim **1**, wherein at least one said checking includes testing if certain functions have already been called, testing if certain functions have been already called with certain parameters.

10. The method according to claim **1** wherein said system function is a random value.

11. The method according to claim 1 wherein said system function is a time value.

12. The method according to claim 1 wherein said system function is a certain folder name.

13. The method according to claim 1 wherein said system function is a number of files in a folder.

14. The method according to claim 1 wherein said system function is a certain folder extension.

15. The method according to claim 1, wherein said system function is a function written in a programming language of the executable.

16. The method according to claim 1, wherein said system function is an operating system function of an operating system under which said code is to be executed.

17. The method according to claim 1, wherein at least one said parameter is a permanent parameter.

18. The method according to claim 1, wherein at least one said parameter is a temporary parameter.

19. The method according to claim 1, wherein at least one said parameter is passed by its content.

20. The method according to claim 1, wherein at least one said parameter is passed by its address.

21. A method for emulating, said method comprising:

providing non-compiled code;

pre-defining at least one content attribute having relevance to maliciousness, for at least one variable of said executable code;

pre-defining at least one malicious state, at least one said malicious state being a call to a certain system function with at least one said content attribute;

upon emulating an assignment instruction within said non-compiled code, setting the content attributes of the assigned variable according to the assigned content, and the value of the assigned variable;

upon emulating a mathematical operator within said non-compiled code, employing content mathematics; and

upon emulating a function call within said non-compiled code, checking that the current state, being the combination of the function identity with the content and/or value of the calling parameters, corresponds to at least one of the pre-defined malicious states, thereby indicating the maliciousness of said code.

22. A method for emulating, said method comprising:

providing non-compiled executable code;

pre-defining at least one content attribute having relevance to maliciousness, for at least one variable of said executable code;

pre-defining at least one malicious state, being a call to a certain system function with certain content and/or value;

upon emulating an assignment instruction within said non-compiled executable code, setting the content attributes of the assigned variable according to the assigned content, and the value of the assigned variable;

upon emulating a mathematical operator within said non-compiled executable code, employing content mathematics; and

upon emulating a function call within said non-compiled executable code, checking that the current state, being the combination of the function identity with the content and/or value of the calling parameters, corresponds to at least one of the pre-defined malicious states, thereby indicating the maliciousness of said code.

23. A method for emulating, said method comprising:

pre-defining at least one content attribute having relevance to maliciousness, for at least one variable of said executable code;

pre-defining at least one malicious state, said at least one said malicious state being a call to a certain system function with a variable having a certain content attribute;

upon emulating an assignment instruction, setting the content attributes of the assigned variable according to the assigned content, and the value of the assigned variable;

upon emulating a mathematical operator, employing content mathematics; and

upon emulating a function call, checking that a first current state, being the combination of the function identity with the content attributes of the calling parameters, corresponds to at least one of the pre-defined malicious states, thereby indicating the maliciousness of said code.

24. The method of claim 23 wherein upon emulating a function call a second current state being the combination of the function identity with the content and/or value of the calling parameters is checked.

* * * * *

Exhibit B

US007386884B2

(12) **United States Patent**
Cohen et al.

(10) **Patent No.: US 7,386,884 B2**
(45) **Date of Patent: Jun. 10, 2008**

(54) **METHOD FOR PREVENTING ACTIVATION OF MALICIOUS OBJECTS**

(75) Inventors: **Oded Cohen**, Tivon (IL); **Yanki Margalit**, Ramat-Gan (IL); **Dany Margalit**, Ramat-Gan (IL)

(73) Assignee: **Aladdin Knowledge Systems Ltd.**, Petach Tikva (IL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 449 days.

(21) Appl. No.: **10/826,503**

(22) Filed: **Apr. 19, 2004**

(65) **Prior Publication Data**

US 2005/0235160 A1 Oct. 20, 2005

(51) **Int. Cl.**
G06F 12/14 (2006.01)
(52) **U.S. Cl.** **726/22**; 713/161
(58) **Field of Classification Search** 726/22–23; 713/161
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,088,803 A *	7/2000	Tso et al.	....	726/22
2004/0054928 A1 *	3/2004	Hall	....	713/201

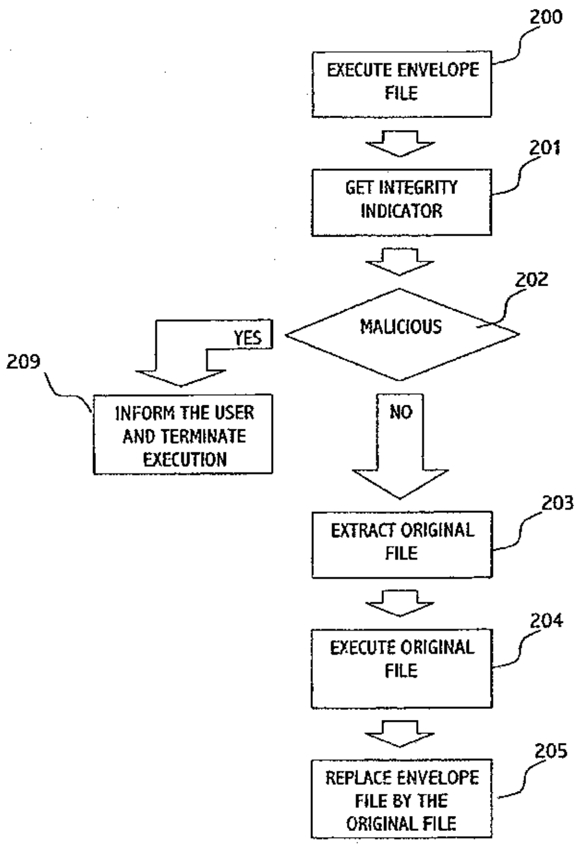
* cited by examiner

Primary Examiner—Gilberto Barron, Jr.
Assistant Examiner—Jacob Lipman
(74) *Attorney, Agent, or Firm*—Mark M. Friedman

(57) **ABSTRACT**

A method for preventing activating a malicious object passing through a checkpoint, and decreasing the overall inspection delay thereof, the method comprising the steps of: (a) at the checkpoint, creating an envelope file, being an executable file comprising: the object; code for extracting the object from the envelope file; and an indicator for indicating the integrity of the object; (b) forwarding the envelope file instead of the object toward its destination, while holding at least a part of the envelope file which comprises the indicator; (c) inspecting the object; and (d) setting the indicator on the envelope file to indicate the inspection result thereof, and releasing the rest of the envelope file.

10 Claims, 4 Drawing Sheets



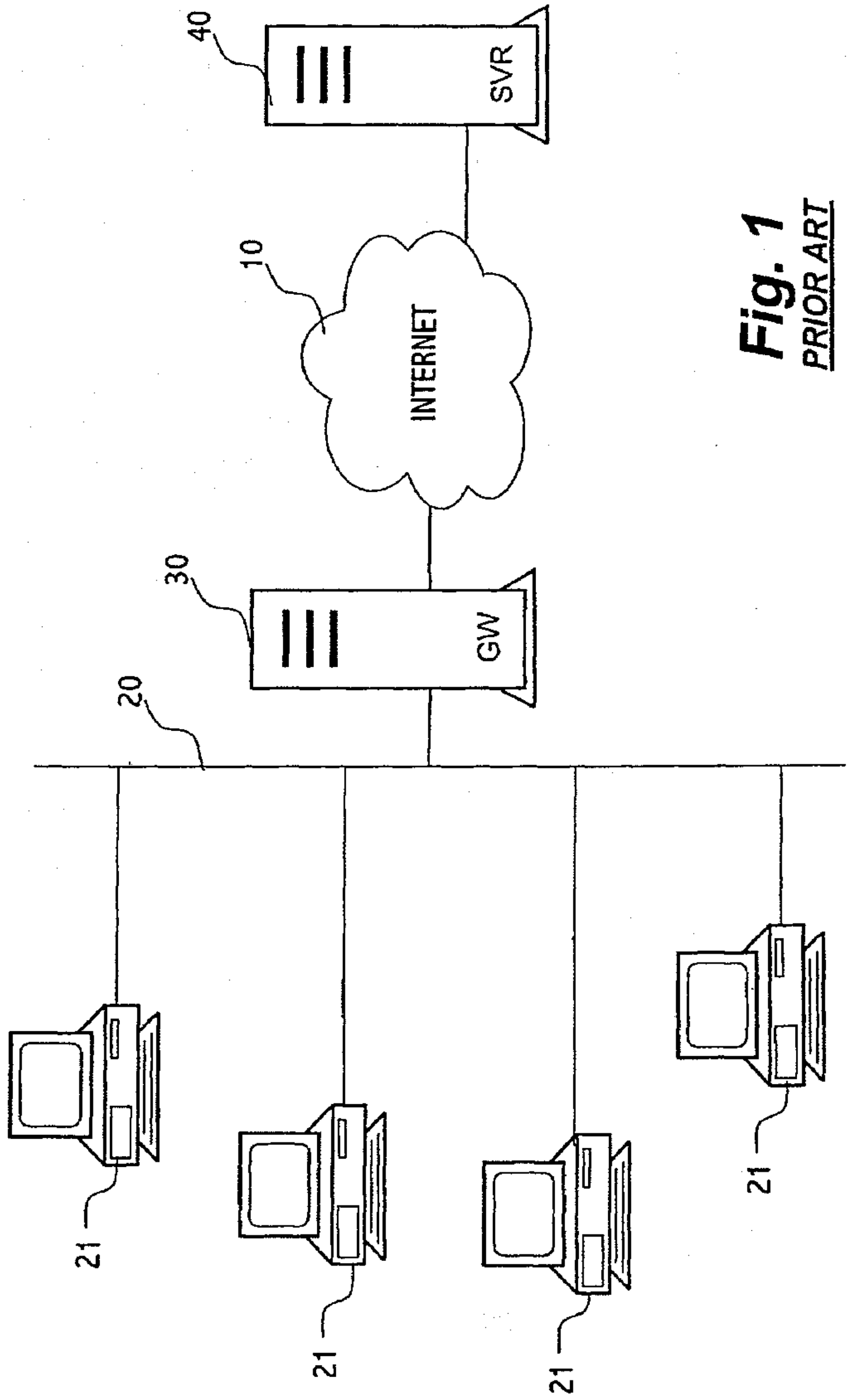


Fig. 1
PRIOR ART

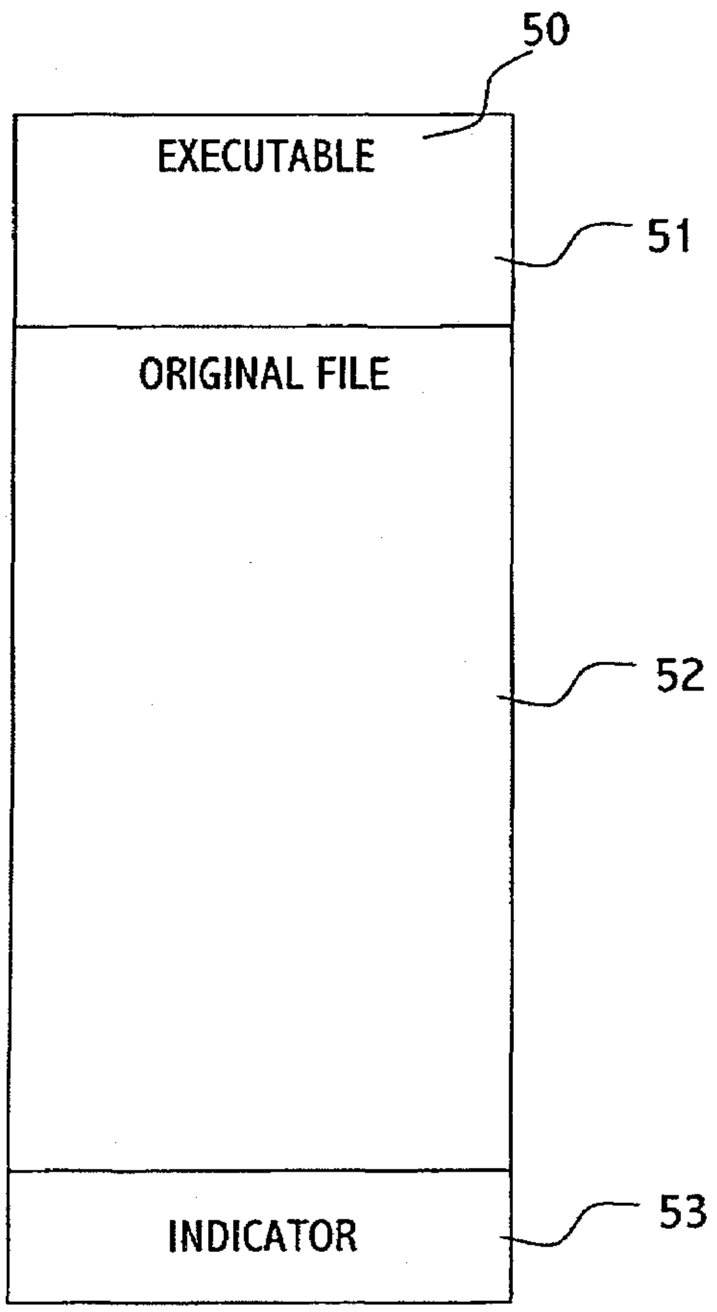


Fig. 2

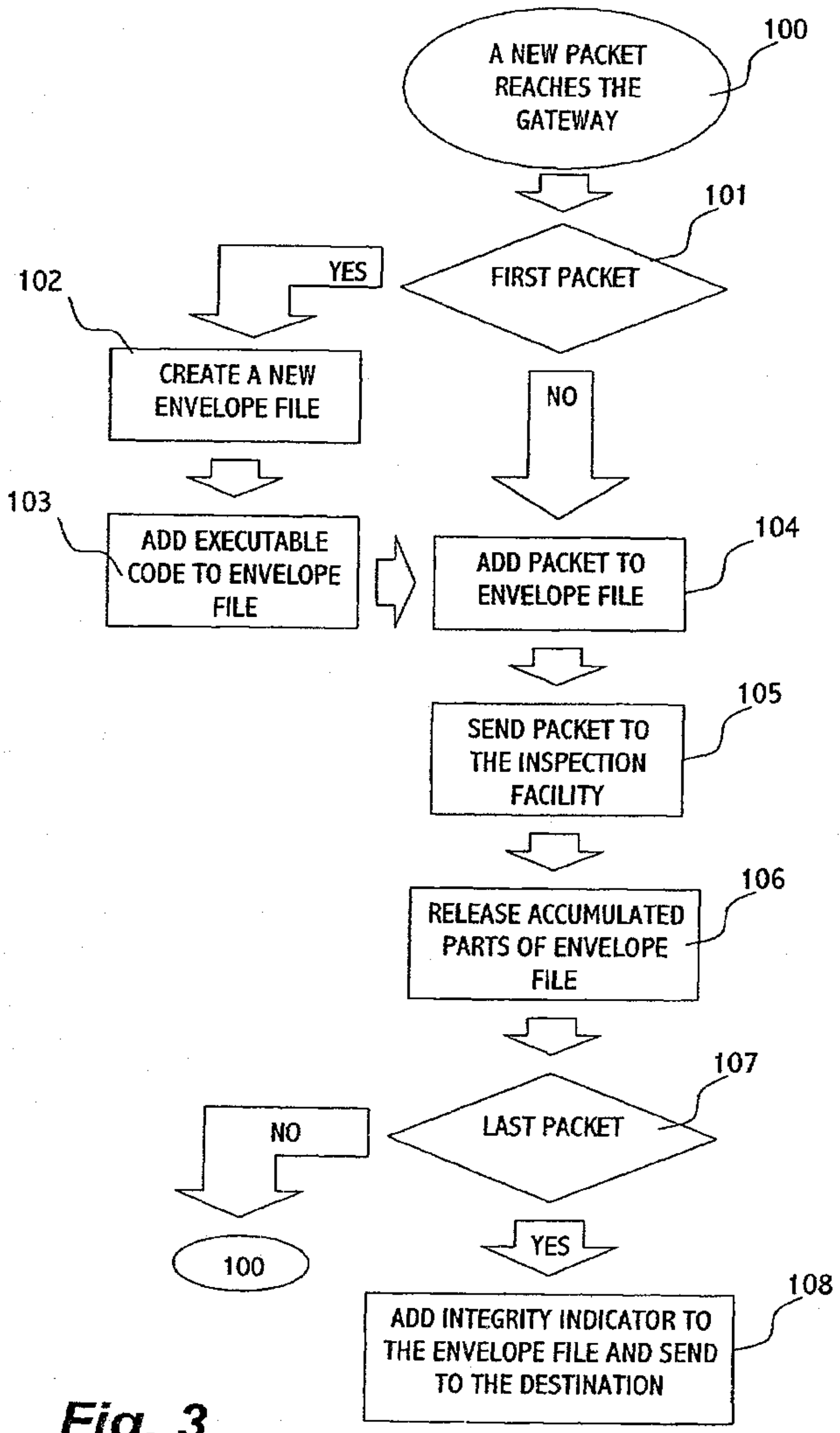

A NEW PACKET REACHES THE GATEWAY
100
FIRST PACKET
101
YES
102
CREATE A NEW ENVELOPE FILE
NO
103
ADD EXECUTABLE CODE TO ENVELOPE FILE
ADD PACKET TO ENVELOPE FILE
104
SEND PACKET TO THE INSPECTION FACILITY
105
RELEASE ACCUMULATED PARTS OF ENVELOPE FILE
106
LAST PACKET
107
NO
100
YES
ADD INTEGRITY INDICATOR TO THE ENVELOPE FILE AND SEND TO THE DESTINATION
108

Fig. 3

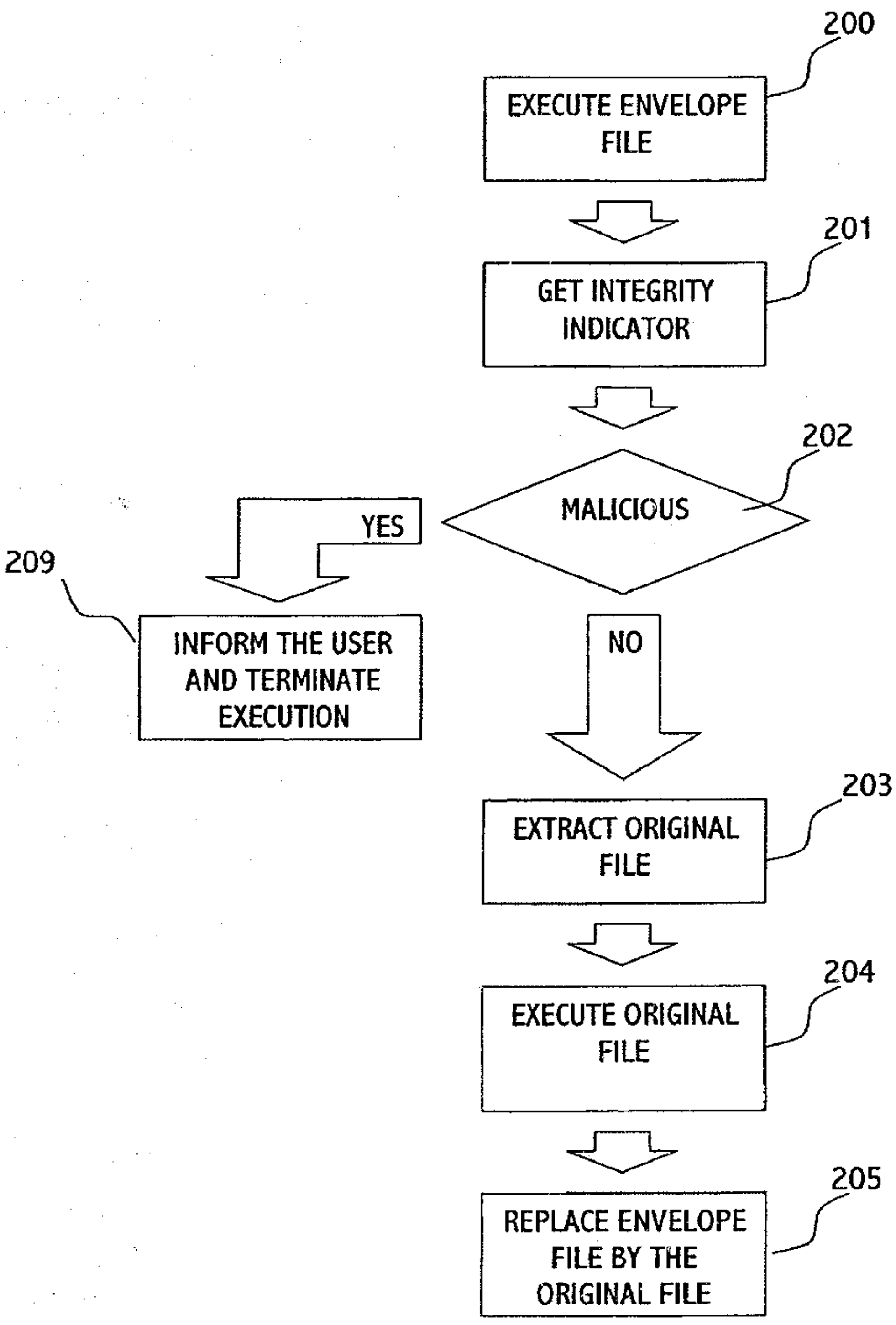

200
EXECUTE ENVELOPE FILE
201
GET INTEGRITY INDICATOR
202
MALICIOUS
YES
209
INFORM THE USER AND TERMINATE EXECUTION
NO
203
EXTRACT ORIGINAL FILE
204
EXECUTE ORIGINAL FILE
205
REPLACE ENVELOPE FILE BY THE ORIGINAL FILE

Fig. 4

METHOD FOR PREVENTING ACTIVATION OF MALICIOUS OBJECTS

FIELD OF THE INVENTION

The present invention relates to the field of detecting viruses and other malicious forms in a checkpoint (e.g. gateway). More particularly, the invention relates to a method for preventing activation of malicious objects while decreasing the overall inspection time.

BACKGROUND OF THE INVENTION

The term "gateway" refers in the art to a bridge between two networks. For each network, the gateway is a point that acts as an entrance to another network. From the implementation point of view, a gateway is often associated with both a router, which knows where to direct a given packet that arrives to the gateway and a switch, which provides the packet with the actual path in and out of the gateway. Due to its nature, the gateway to a local network is a proper point for checking out objects (e.g. files and email messages) that pass through it, in order to detect viruses and other forms of maliciousness ("inspection") before reaching the user.

As a filtering facility, a gateway server has to deal with two contradicting objects: on the one hand, it has to hold a file that reaches the gateway in its path from a source to a destination until the inspection indicates it is harmless and thereby prevents its execution on the destination site, on the other hand holding a file at the gateway server until the inspection process terminates which results in a bottleneck to data traffic passing through the gateway.

Inspection activity has a substantial influence on the traffic speed through a gateway. U.S. patent application Ser. No. 10/002,407, titled as Security Router, deals with this problem by skipping the inspection of trusted files. According to this invention, since multimedia files (e.g. JPG files) do not comprise executable code (according to their definition), these files are not inspected, thereby diminishing the delay caused by the inspection process.

U.S. patent application Ser. No. 09/498,093, titled as "Protection of computer networks against malicious content", deals with this problem by holding in a checkpoint (e.g. a gateway) only a part of the file, such as the last packet of the file, and releasing it once the file has been indicated as harmless. This way the majority of the file is not delayed at the gateway, but its execution at the destination site cannot be carried out until the last part reaches the destination. This solution is applicable only for files that in order to be executed or activated, the whole file has to be available on the executing platform. However, if the executing platform activates a file even in the case where only a part of the file is available, the executing platform is exposed to viruses and other malicious forms.

Furthermore, some inspection methods, such as CRC-based methods, require that the whole file be available during the inspection process. Files that should be fully accessible for inspection, may cause a substantial delay to the traffic through a checkpoint since the inspection can start only after the whole file is accessible to the inspection facility. Thus, in this case, the parts of a file should be accumulated and held at the inspection point until the inspection indicates that it is harmless, and only then the file may be "released" to its destination.

It is an object of the present invention to provide a method for preventing activation of malicious objects.

It is a further object of the present invention to provide a method for preventing from a checkpoint the activation of malicious objects on the executing platform.

It is a still further object of the present invention to provide a method for inspecting a file on a checkpoint, by which the delay thereof is decreased in comparable to the prior art.

Other objects and advantages of the invention will become apparent as the description proceeds.

SUMMARY OF THE INVENTION

A method for preventing activating a malicious object passing through a checkpoint, and decreasing the overall inspection delay thereof, the method comprising the steps of: (a) at the checkpoint, creating an envelope file, being an executable file comprising: the object; code for extracting the object from the envelope file; and an indicator for indicating the integrity of the object; (b) forwarding the envelope file instead of the object toward its destination, while holding at least a part of the envelope file which comprises the indicator; (c) inspecting the object; and (d) setting the indicator on the envelope file to indicate the inspection result thereof, and releasing the rest of the envelope file.

According to a preferred embodiment of the invention, the envelope file is an auto-executable file. According to one embodiment of the invention, the name of the envelope file is identical to the name of the inspected object. According to another embodiment of the invention, the name of the envelope file differs than the name of the inspected object.

The indicator may be a CRC of a part of the envelope file, a CRC of a part of the inspected object, a checksum of a part of the envelope file, a checksum of a part of the inspected object, a value stored within the envelope file, absence of a part of the envelope file, absence of a part of the inspected object, and so forth.

According to a preferred embodiment of the invention, the envelope file further comprises code for displaying an acknowledgment to the user for indicating the integrity of the inspected object. Typically the acknowledgment is displayed only when the inspected objects is indicated as malicious.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood in conjunction with the following figures:

FIG. **1** schematically illustrates a system that may be used for implementing the present invention.

FIG. **2** schematically illustrates the parts of an envelope file, according to a preferred embodiment of the present invention.

FIG. **3** is a flowchart of a process for creating an envelope file, according to a preferred embodiment of the invention.

FIG. **4** is a flowchart of a process carried out by an envelope file on the destination site, according to a preferred embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The term "checkpoint" refers herein to a point on a data channel or data junction in which the passing data is inspected. For example, a gateway is a suitable place for a checkpoint.

FIG. 1 schematically illustrates a system that may be used for implementing the present invention. The computers **21** are connected to the local area network **20**. The local area network **20** is connected to the internet **10**. The gateway server **30** is interposed between the local area network **20** and the internet **10**. The Internet server **40** hosts web sites. A browser being executed on a computer **21** that addresses to the web site hosted by the internet server **40** cause files to be transferred from the internet server **40** to the computer **21** through the gateway server **30**.

As mentioned above, a gateway server that inspects data that is transferred through it, has to deal with two contradicting objects: on the one hand, it has to hold an inspected object until its harmlessness is indicated, on the other hand holding the file at the gateway may cause a bottleneck to the data traffic passing through the gateway.

This problem is solved by the present invention. The present invention allows passing a file toward its destination while inspecting the file, and still preventing the activation of the file at the destination site until its integrity is indicated. This is carried out as follows: instead of forwarding the original file to its destination, a substitute file which comprises the original file is constructed and transmitted to the destination. The substitute file is also referred herein as envelope file.

According to one embodiment of the invention, the envelope file is an executable which comprises the following parts:

the original file;

an indicator about the integrity of the original file; and

an executable part, upon which execution extracts the original file, and executes it.

At the destination site, the envelope file is executed instead of the original file. On this execution, if the integrity indicator indicates that the original file is harmless, the original file is extracted from the envelope file and executed.

The integrity indicator can be the CRC (Cyclic Redundancy Checks) or checksum of the envelope file, the CRC or checksum of the original file, a bit within the envelope file, a value stored within the envelope file (in a byte, a bit, etc.), and so forth. In order to prevent forge of the envelope file, the indicator and/or the envelope file can be secured (e.g. encrypted, digitally signed with the original or envelope file, and so forth).

For example, in the case where the integrity of the original file is indicated at the gateway, the gateway sends the true CRC value of the original file. In case the original file is indicated as comprising malicious content, the gateway sends a wrong CRC value of the original file. As well, other indicators can be used.

It should be noted that the envelope file performs two operations: on the one hand it is an executable file, which is loaded into the computer's memory for executing, but on the other hand it is treated as a data file.

FIG. 2 schematically illustrates the parts of an envelope file, according to a preferred embodiment of the present invention. The envelope file **50** comprises:

an executable part **51**;

the original file **52**; and

an integrity indicator **53**.

FIG. 3 is a flowchart of a process for creating an envelope file, according to a preferred embodiment of the invention.

The process starts at block **100**, where a packet of a file sent from a source to a destination reaches to a gateway server (or, generally, a checkpoint).

From block **101**, if the packet is the first packet of a file then the process continues with block **102**, where a new

envelope file is created, and therefrom with block **103**, where an executable code is added to the created envelope file, and therefrom to block **104**. The operation of the executable code is detailed in the description of FIG. 4.

At block **104**, the packet is added to the envelope file.

At block **105**, a copy of the packet is sent to the inspection facility for inspection.

At block **106**, the accumulated parts of the envelope file are sent to the destination.

From block **107**, if the reached packet is the last packet of the file, then an integrity indicator is added to the envelope file, and the last accumulated parts of the envelope file are sent to the destination. The integrity indicator is received from the inspection facility which operates at the gateway. If the received packet is not the last packet of the original file, then when a new packet of the original file will reach to the gateway server, the process will continue from block **100**.

In this example it was assumed that the packets of a file reach the gateway in consecutive order, however, as known to a person of ordinary skill in the art, this is not necessarily true, since each packet may be redirected to its destination through a different path. Thus, when adding a packet of the original file to the envelope file, the order should be taken into consideration. However, according to a preferred embodiment of the invention, the indicator is stored within the last part of the envelope file. This way usually (but not always) the integrity indicator will be the last to reach the destination.

FIG. **4** is a flowchart of a process carried out by an envelope file on the destination site, according to a preferred embodiment of the invention.

At block **200** the envelope file is executed.

At block **201**, the executed envelope program gets the integrity indicator. As mentioned above, the envelope file is also used as a data file which comprises the original file and an integrity indicator.

From block **202**, if according to the integrity indicator the original file is malicious, then at block **209** the envelope file acknowledges the user (e.g. by displaying a corresponding message), and the execution of the envelope file terminates, otherwise control continues with block **203**.

At block **203**, the envelope program extracts the original file from the envelope file.

At block **204**, the original file is executed. At this point the envelope executable can terminate.

At block **205**, since the envelope file is not required anymore, it can be replaced by the original file.

Typically the inspected object is an executable, however it should be noted that the invention can also be implemented for other file types, whether it is an executable or not, or whether it is attached to an email message or returned in an FTP or HTTP session.

It should also be noted that although the references herein are to a gateway, the invention can be implemented to any kind of checkpoint, e.g. an arbitrary point on a network, a server of an Internet service provider, a mail server, and so forth.

Those skilled in the art will appreciate that the invention can be embodied by other forms and ways, without losing the scope of the invention. The embodiments described herein should be considered as illustrative and not restrictive.

The invention claimed is:

1. A method for preventing activating a malicious object passing through a checkpoint, and decreasing the overall inspection delay thereof, the method comprising the steps of:

a. at said checkpoint, creating an envelope file, being an executable file comprising: said object; code for extracting said object from said envelope file; and an indicator for indicating the integrity of said object;

b. forwarding said envelope file instead of said object toward its destination, while holding at least a part of said envelope file which comprises said indicator;

c. inspecting said object;

d. setting said indicator on said envelope file to indicate the inspection result thereof, and

e. releasing the rest of said envelope file.

2. A method according to claim **1**, wherein said checkpoint is selected from a group comprising: a gateway server, a proxy server.

3. A method according to claim **1**, wherein the name of said envelope file is identical to the name of the inspected object.

4. A method according to claim **1**, wherein the name of said envelope file differs than the name of the inspected object.

5. A method according to claim **1**, wherein said indicator is selected from a group comprising: a CRC of at least one part of said envelope file, a CRC of at least one part of said inspected object, a checksum of at least one part of said envelope file, a checksum of at least one part of said inspected object, a value stored within said envelope file, absence of a part of said envelope file, absence of a part of said object.

6. A method according to claim **1**, wherein at least a part of said object is secured.

7. A method according to claim **1**, wherein at least a part of said envelope file is secured.

8. A method according to claim **1**, wherein said indicator is stored within the last part of said envelope file.

9. A method according to claim **1**, wherein said envelope file further comprises code for displaying an acknowledgment.

10. A method according to claim **9**, wherein said acknowledgment indicates integrity of said object.

* * * * *